МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

**Курсовая работа**

**по дисциплине «программирование»**

**Тема: «Класс - структура данных в двоичном файле**»

Факультет: АВТФ Преподаватель: Менжулин С.А.

Группа: АВТ-342

Студенты: Фадеев В.А.

Вариант: 4.2

Новосибирск – 2024

Оглавление

[**1.** **ОПИСАНИЕ ЗАДАНИЯ** 3](#_Toc123078659)

[**2.** **СТРУКТУРНОЕ ОПИСАНИЕ РАЗРАБОТКИ** 4](#_Toc123078660)

[**3.** **ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТКИ** 6](#_Toc123078661)

[**4.** **ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА** 13](#_Toc123078662)

[**5.** **ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ НА КОНТРОЛЬНЫХ ПРИМЕРАХ** 18](#_Toc123078663)

[ВЫВОДЫ 19](#_Toc123078664)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 20](#_Toc123078665)

[ПРИЛОЖЕНИЕ 21](#_Toc123078666)

1. **ОПИСАНИЕ ЗАДАНИЯ**

Класс двоичного файла, производный от fstream. Двоичный файл содержит заданную структуру данных. Программа должна представлять собой простой текстовый редактор, использующий структуру данных для промежуточного хранения редактируемого файла. Должны быть реализованы операции создания и заполнения двоичного файла из заданного текстового и сохранения содержимого двоичного файла в текстовый.

Программа должна реализовывать указанные выше действия. Протестировать структуру данных. Программа тестирования должна содержать **меню**, обеспечивающее выбор операций.

Вариант 4.2.

Класс – двоичный файл, производный от fstream. Файл содержит односвязный список строк в формате записей переменной длины со счетчиком. Формат файла: в начале файла – заголовок списка. Элемент списка содержит файловый указатель на следующий и саму строку в виде записи переменной длины.

1. **СТРУКТУРНОЕ ОПИСАНИЕ РАЗРАБОТКИ**

Для реализации программы, которая работает с двусвязным списком, хранящимся в бинарном файле, был разработан класс LinkedFile. Класс организует управление списком через бинарный файл, включая добавление, удаление, сортировку и загрузку/сохранение данных.

#### **Внутренняя структура данных**

Бинарный файл состоит из следующих компонентов:

1. **Заголовок файла (Header):**
   * head — позиция первой записи в списке.
   * count — количество записей в списке.
2. **Элементы списка (Node):**  
   Каждый элемент содержит:
   * next — позицию следующего элемента.
   * length — длину строки в байтах.
   * Данные строки (динамический массив символов).

#### **Описание класса LinkedFile:**

Класс LinkedFile наследуется от std::fstream и предоставляет интерфейс для работы с бинарным файлом как с двусвязным списком.

**Основные поля класса:**

* Header header — структура, хранящая заголовок файла.
* string filename — имя файла для работы с данными.

**Методы класса:**

1. **Внутренние функции работы с заголовком:**
   * void writeHeader() — запись заголовка в бинарный файл.
   * void readHeader() — чтение заголовка из бинарного файла.
2. **Работа с элементами списка:**
   * void add(const std::string& str) — добавление строки в конец списка.
   * void insertAt(int index, const std::string& str) — вставка строки на заданную позицию по индексу.
   * void deleteNode(int index) — удаление элемента по индексу.
   * void getStringAt(int index) — получение строки по индексу.
   * void clear() — удаление всех элементов списка.
3. **Сортировка:**
   * void shellSort() — реализация сортировки Шелла для элементов списка (сортирует по возрастанию длины строк в узлах).
4. **Работа с файлами:**
   * void loadFromTextFile(const string &textFilename) — загрузка данных из текстового файла.
   * void saveToTextFile(const string &textFilename) — сохранение данных в текстовый файл.
   * void saveToBinaryFile(const string &binaryFilename) — сохранение данных в бинарный файл.
   * void loadFromBinaryFile(const string &binaryFilename) — загрузка данных из бинарного файла.
5. **Отображение содержимого:**
   * void display() — вывод содержимого списка на экран.

#### Логика работы

1. **Создание экземпляра класса:**  
   При создании объекта LinkedFile передаётся имя файла. Если файл отсутствует, он создаётся с пустым заголовком.
2. **Добавление и управление элементами:**  
   Методы add и insertAt записывают новые строки в файл, обновляя указатели и заголовок.
3. **Сортировка:**  
   Метод shellSort производит сортировку строк в файле, работая напрямую с элементами через указатели.
4. **Импорт/экспорт данных:**  
   Методы loadFromTextFile и saveToTextFile обеспечивают взаимодействие с текстовыми файлами, а saveToBinaryFile и loadFromBinaryFile позволяют сохранять и загружать данные из бинарных файлов.
5. **ФУНКЦИОНАЛЬНОЕ ОПИСАНИЕ РАЗРАБОТКИ**

* Конструктор по умолчанию для сохранения и загрузки бинарных файлов:

LinkedFile::LinkedFile(const string& filename) : filename(filename) {  
 this->open(filename, ios::in | ios::out | ios::binary);  
 if (!this->is\_open()) {  
 this->open(filename, ios::out | ios::binary);  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
 } else {  
 readHeader();  
 }  
}

 **Инициализация переменных:**

* Конструктор принимает имя файла filename и сохраняет его в поле filename объекта.

 **Попытка открыть файл:**

* this->open(filename, ios::in | ios::out | ios::binary) — открывается файл для чтения и записи в бинарном режиме.
* Если файл не существует или не удаётся его открыть, метод is\_open() вернёт false.

 **Создание нового файла:**

* Если файл отсутствует:
  + this->open(filename, ios::out | ios::binary) — открывается файл в режиме записи, чтобы его создать.
  + Инициализируются поля заголовка:
    - header.head = -1; — указывает, что список пуст (нет указателя на первую запись).
    - header.count = 0; — указывает, что элементов в списке пока нет.
  + writeHeader(); — заголовок записывается в файл.
  + this->close(); — закрывается созданный файл.
  + Затем файл открывается снова в режиме чтения и записи: this->open(filename, ios::in | ios::out | ios::binary).

 **Чтение заголовка из существующего файла:**

* Если файл уже существует и успешно открыт, вызывается метод readHeader(), который считывает заголовок, чтобы получить текущую информацию о списке (указатель на первую запись и количество элементов).

### **writeHeader — Запись заголовка в файл**

// Запись заголовка в начало файла  
void LinkedFile::writeHeader() {  
 this->seekp(0, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&header.head), sizeof(header.head));  
 this->write(reinterpret\_cast<char\*>(&header.count), sizeof(header.count));  
}

Этот метод записывает заголовок (состояние списка) в начало бинарного файла.

1. **Перемещение в начало файла:**
   * this->seekp(0, ios::beg); перемещает указатель записи на начало файла.
2. **Запись заголовка:**
   * header.head (позиция первого узла в файле) записывается как бинарные данные.
   * header.count (количество узлов в списке) также записывается как бинарные данные.

### **readHeader — Чтение заголовка из файла**

// Чтение заголовка из файла  
void LinkedFile::readHeader() {  
 this->seekg(0, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&header.head), sizeof(header.head));  
 this->read(reinterpret\_cast<char\*>(&header.count), sizeof(header.count));  
}

Этот метод загружает заголовок из бинарного файла в объект.

1. **Перемещение в начало файла:**
   * this->seekg(0, ios::beg); перемещает указатель чтения на начало файла.
2. **Чтение заголовка:**
   * Заголовок читается в поля header.head и header.count.

### **add — Добавление строки в конец списка**

void LinkedFile::add(const string& str) {  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
  
 // Переход в конец файла  
 this->seekp(0, end);  
 streampos currentPos = this->tellp();  
  
 // Запись нового узла  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 // Обновление предыдущего узла, если он есть  
 if (header.head == -1) {  
 header.head = currentPos;  
 } else {  
 streampos prevPos = header.head;  
 Node prevNode;  
  
 while (true) {  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
 if (prevNode.next == -1) {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentPos), sizeof(currentPos));  
 break;  
 }  
 prevPos = prevNode.next;  
 }  
 }  
  
 header.count++;  
 writeHeader();  
}

Этот метод добавляет новую строку в конец двусвязного списка.

1. **Создание нового узла:**
   * newNode.length — длина строки.
   * newNode.next — указатель на следующий узел (изначально -1, так как узел последний).
2. **Запись узла в конец файла:**
   * Переход к концу файла (seekp(0, end)).
   * Запись структуры newNode и самой строки.
3. **Обновление связей:**
   * Если список пуст (т.е. header.head == -1), новый узел становится первым.
   * Если список не пуст, обновляется указатель next предыдущего узла.
4. **Обновление заголовка:**
   * Увеличивается счётчик элементов в списке (header.count++).
   * Заголовок перезаписывается (writeHeader()).

### **display — Отображение всех строк**

// Отображение всех строк  
void LinkedFile::display() {  
 if (header.head == -1) {  
 cout << "List is empty.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
 int counter = 0;  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 counter++;  
 cout << counter << ": " << str << "\n";  
  
 currentPos = node.next;  
 }  
}

Этот метод последовательно считывает и выводит строки из файла.

1. **Проверка пустого списка:**
   * Если header.head == -1, выводится сообщение, что список пуст.
2. **Чтение узлов:**
   * Считываются данные узлов по цепочке указателей next.
3. **Вывод данных:**
   * Каждая строка отображается в консоли с её порядковым номером.

### **insertAt — Вставка строки по индексу**

// Вставка строки по индексу  
void LinkedFile::insertAt(int index, const string& str) {  
 if (cin.fail()) {  
 throw invalid\_argument("Invalid input");  
 }  
 if (index < 0) {  
 throw out\_of\_range("Index cannot be negative.");  
 }  
 if (index > header.count) {  
 throw out\_of\_range("Index is out of range.");  
 }  
  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
  
 // Определение позиции нового узла  
 streampos newPos = this->tellp();  
 this->seekp(0, ios::end);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 if (header.count == 0) {  
 // Вставка в пустой список  
 header.head = newPos;  
 cout << "Inserted first node.\n";  
 } else if (index == 0) {  
 // Вставка в начало списка  
 newNode.next = header.head;  
 header.head = newPos;  
 this->seekp(newPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 cout << "Inserted node at the beginning.\n";  
 } else {  
 // Вставка в середину или конец списка  
 streampos prevPos = header.head;  
 Node prevNode;  
  
 for (int i = 0; i < index - 1; i++) {  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
 prevPos = prevNode.next;  
 }  
  
 // Считывание предыдущего узла  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
  
 newNode.next = prevNode.next;  
 prevNode.next = newPos;  
  
 // Запись обновлённого предыдущего узла  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
  
 // Запись нового узла  
 this->seekp(newPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
  
 cout << "Inserted node at index " << index << ".\n";  
 }  
  
 header.count++;  
 writeHeader();  
}

Этот метод вставляет новую строку на заданную позицию.

1. **Проверка индекса:**
   * Если индекс выходит за границы списка, генерируется исключение.
2. **Создание нового узла:**
   * Запись нового узла и строки в конец файла.
3. **Обновление связей:**
   * Если вставка в начало (index == 0), новый узел становится первым.
   * Если вставка в середину, предыдущий узел обновляет указатель next на новый узел.
4. **Обновление заголовка:**
   * Увеличивается счётчик элементов.

### **deleteNode — Удаление узла по индексу**

void LinkedFile::deleteNode(int index) {  
 if (index < 0 || index >= header.count) {  
 cout << "Out of bounds.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 streampos prevPos = -1;  
 Node currentNode;  
  
 for (int i = 0; i <= index; i++) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&currentNode.next), sizeof(currentNode.next));  
 this->read(reinterpret\_cast<char\*>(&currentNode.length), sizeof(currentNode.length));  
  
 if (i == index) {  
 if (prevPos == -1) {  
 header.head = currentNode.next;  
 } else {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentNode.next), sizeof(currentNode.next));  
 }  
  
 header.count--;  
 writeHeader();  
 cout << "Node deleted.\n";  
 return;  
 }  
  
 prevPos = currentPos;  
 currentPos = currentNode.next;  
 }  
}

Этот метод удаляет узел с заданным индексом.

1. **Проверка индекса:**
   * Если индекс выходит за границы списка, выводится сообщение.
2. **Поиск узла:**
   * Узел и его предыдущий узел определяются путём последовательного обхода.
3. **Обновление связей:**
   * Если удаляется первый узел, header.head обновляется на next удаляемого узла.
   * Если узел в середине/конце, next предыдущего узла обновляется на next удаляемого.
4. **Обновление заголовка:**
   * Счётчик элементов уменьшается.

### **getStringAt — Получение строки по индексу**

void LinkedFile::getStringAt(int index) {  
 if (index < 0 || index >= header.count) {  
 cout << "Out of bounds.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
  
 for (int i = 0; i <= index; i++) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 if (i == index) {  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 cout << "String at index " << index << ": " << str << "\n";  
 return;  
 }  
  
 currentPos = node.next;  
 }  
}

Этот метод считывает и выводит строку с заданным индексом.

1. **Проверка индекса:**
   * Если индекс выходит за границы списка, выводится сообщение.
2. **Последовательный обход:**
   * Узел считывается, пока не достигнется нужный индекс.
3. **Вывод строки:**
   * Строка с заданным индексом выводится в консоль.

### **clear — Очистка списка**

void LinkedFile::clear() {  
 this->close();  
 this->open(filename, ios::out | ios::binary);  
 if (this->is\_open()) {  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 cout << "File and list cleared.\n";  
 } else {  
 cout << "Failed to clear the file.\n";  
 }  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
}

### **shellSort — Сортировка списка по длине строк**

void LinkedFile::shellSort() {  
 if (header.count <= 1) {  
 cout << "List is already sorted or empty.\n";  
 return;  
 }  
  
 // Считывание всех узлов в память  
 vector<string> strings; // Хранение строк  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 strings.push\_back(str);  
  
 currentPos = node.next;  
 }  
 int n = strings.size();  
 for (int gap = n / 2; gap > 0; gap /= 2) {  
 for (int i = gap; i < n; i++) {  
 string temp = strings[i];  
 int j;  
 for (j = i; j >= gap && strings[j - gap].size() > temp.size(); j -= gap) {  
 strings[j] = strings[j - gap];  
 }  
 strings[j] = temp;  
 }  
 }  
 this->close();  
 this->open(filename, ios::out | ios::binary);  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 streampos prevPos = -1;  
 for (const auto& str : strings) {  
 // Позиция текущего узла  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
 this->seekp(0, ios::end);  
 streampos currentPos = this->tellp();  
 // Если первый узел, обновляем заголовок  
 if (header.head == -1) {  
 header.head = currentPos;  
 }  
  
 // Связывание предыдущего узла с текущим  
 if (prevPos != -1) {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentPos), sizeof(currentPos));  
 }  
  
 // Запись текущего узла  
 this->seekp(currentPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 prevPos = currentPos;  
 header.count++;  
 }  
 writeHeader();  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
 cout << "List has been sorted by string length.\n";  
}

Этот метод сортирует строки по их длине с использованием алгоритма Шелла.

1. **Считывание всех данных в память:**
   * Узлы последовательно считываются, а строки сохраняются в векторе.
2. **Алгоритм Шелла:**
   * Строки сортируются по длине в векторе.
3. **Перезапись файла:**
   * Файл очищается, и строки записываются заново в отсортированном порядке.

### **loadFromTextFile — Загрузка из текстового файла**

Этот метод добавляет строки из текстового файла в бинарный список.

// Загрузка списка из текстового файла  
void LinkedFile::loadFromTextFile(const string& textFilename) {  
 ifstream inFile(textFilename, ios::in);  
  
 if (!inFile) {  
 cerr << "Error: Unable to open file " << textFilename << endl;  
 return;  
 }  
 string fileContent((istreambuf\_iterator<char>(inFile)), istreambuf\_iterator<char>());  
 inFile.close();  
 string currentLine;  
 for (char c : fileContent) {  
 if (c == '\n') {  
 add(currentLine);  
 currentLine.clear();  
 } else {  
 currentLine += c;  
 }  
 }  
 if (!currentLine.empty()) {  
 add(currentLine);  
 }  
 cout << "File loaded successfully from " << textFilename << endl;  
}

1. **Считывание текста:**
   * Файл считывается построчно.
2. **Добавление в список:**
   * Каждая строка добавляется в список через метод add.

### **saveToTextFile — Сохранение в текстовый файл**

// Сохранение списка в текстовый файл  
void LinkedFile::saveToTextFile(const string& textFilename) {  
 ofstream textFile(textFilename);  
 if (!textFile.is\_open()) {  
 cerr << "Failed to open text file.\n";  
 return;  
 }  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 textFile << str << "\n";  
 currentPos = node.next;  
 }  
 cout << "Saved data from binary file to text file.\n";  
}

Этот метод сохраняет строки списка в текстовый файл.

1. **Последовательное считывание:**
   * Узлы считываются по цепочке, строки извлекаются.
2. **Запись в файл:**
   * Каждая строка записывается в текстовый файл с новой строки.

### **saveToBinaryFile — Сохранение в бинарный файл**

// Сохранение списка в бинарный файл  
void LinkedFile::saveToBinaryFile(const string &binaryFilename) {  
 ofstream outFile(binaryFilename, ios::binary);  
 if (!outFile) {  
 cerr << "Error: Unable to open file for writing: " << binaryFilename << endl;  
 return;  
 }  
 outFile.write(reinterpret\_cast<const char\*>(&header), sizeof(header));  
 streampos currentPos = header.head;  
 while (currentPos != streampos(-1)) {  
 Node node;  
 this->seekg(currentPos);  
 this->read(reinterpret\_cast<char\*>(&node), sizeof(Node));  
 outFile.write(reinterpret\_cast<const char\*>(&node), sizeof(Node));  
 string data(node.length, '\0');  
 this->read(&data[0], node.length);  
 outFile.write(data.c\_str(), node.length);  
 currentPos = node.next;  
 }  
 outFile.close();  
 cout << "Linked list successfully saved to binary file: " << binaryFilename << endl;  
}

Этот метод сохраняет заголовок и все узлы списка в новый бинарный файл.

1. **Запись заголовка:**
   * Поля header записываются в новый файл.
2. **Запись узлов:**
   * Узлы последовательно считываются из списка и записываются в файл.

### **loadFromBinaryFile — Загрузка из бинарного файла**

// Загрузка списка из бинарного файла  
void LinkedFile::loadFromBinaryFile(const string& binaryFilename) {  
 ifstream inFile(binaryFilename, ios::binary);  
 if (!inFile) {  
 cerr << "Error: Unable to open file for reading: " << binaryFilename << endl;  
 return;  
 }  
 clear();  
 while (!inFile.eof()) {  
 size\_t length = 0;  
 inFile.read(reinterpret\_cast<char\*>(&length), sizeof(length));  
 if (inFile.eof()) break;  
 string data(length, '\0');  
 inFile.read(&data[0], length);  
 add(data);  
 }  
 inFile.close();  
 cout << "List successfully loaded from binary file: " << binaryFilename << endl;  
}

Этот метод загружает список из бинарного файла.

1. **Очистка текущего файла:**
   * Список очищается перед загрузкой.
2. **Последовательное считывание:**
   * Узлы и строки считываются из файла и добавляются в список.
3. **ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА**

Для работы с файлом был создан пользовательский интерфейс, позволяющий удобно взаимодействовать с программой. Меню программы (Рис. 1.1) включает в себя следующие пункты:

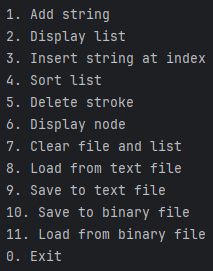


Рис. 1.1 Демонстрация меню программы

Для вызова какого-либо пункта меню необходимо нажать соответствующую клавишу.

Рассмотрим функционал программы по пунктам:

* В п.1 добавляем строки (Рис. 1.2.)



Рис.1.2 Демонстрация ввода строк

Для демонстрации результата дальнейшей работы программы введем несколько строк (Рис. 1.3.):

Рис. 1.3 Исходные строки (строки перед работой)

* В п.2 можем вывести список (Рис. 1.4)

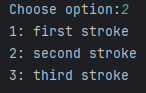


Рис. 1.4 Вывод списка

* В п.3 можем добавить любую строку по индексу, т.е. введя индекс строки и её содержимое (Рис. 1.5.)

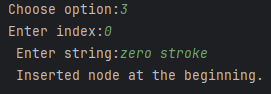
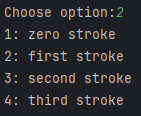
 

Рис.1.5 Демонстрация ввода строки по её индексу

* В п.4 можно отсортировать строки по длине (Рис. 1.6.)

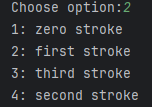
 

Рис. 1.6 Демонстрация сортировки строк по длине

* В п.5 мы можем удалить строку по индексу (Рис. 1.7.)

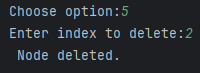
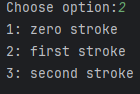
 

Рис. 1.7. Демонстрация удаления элемента списка

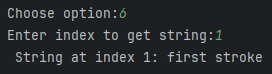
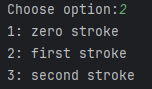
* В п.6 мы можем вывести содержимое конкретного узла (Рис. 1.8.)  
   

Рис. 1.8. Демонстрация вывода одного узла

* В п.7 мы можем очистить весь список и бинарный файл (Рис. 1.9)

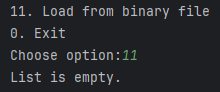
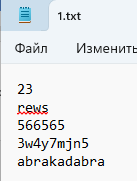
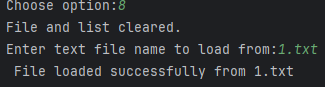
 

Рис. 1.9. Демонстрация удаления списка

* В п.8 мы можем загрузить список из текстового файла (Рис. 1.10)



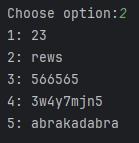
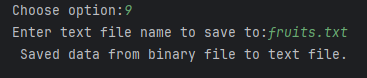


Рис. 1.10. Демонстрация загрузки списка из текстового файла

* В п.9 мы сохранить список в текстовый файл (Рис. 1.11)

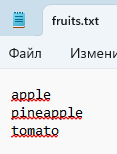


Рис. 1.11. Демонстрация сохранения списка в текстовый файл

* В п.10 мы сохранить список в бинарный файл, в п.11 загрузить из бинарного файла (Рис. 1.12)

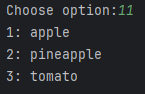
 

Рис. 1.12 Демонстрация сохранения и загрузки из бинарного файла

1. **ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ НА КОНТРОЛЬНЫХ ПРИМЕРАХ**

Для оценки производительности программы проведём эксперимент, используя текстовый файл «data.txt», содержащий 1000 строк. Сначала загрузим строки из файла в двусвязный список, а затем измерим время выполнения операции.

Загрузка строк из текстового файла объёмом 1000 строк: 142 миллисекунды.

Как показано на Рис. 2.1, при увеличении количества записей время выполнения операции заметно возрастает. Таким образом, увеличение объёма данных напрямую влияет на скорость обработки.

Рис 2.1. График зависимости времени работы программы от количества строк

# ВЫВОДЫ

В результате работы была разработана программа для работы с двусвязным списком, хранящимся в бинарном файле. Основной класс программы — LinkedFile, который позволяет эффективно управлять списком с использованием операций чтения, записи и сортировки данных непосредственно в файле.

В ходе работы были реализованы следующие возможности:

* Добавление строки в конец или определённую позицию списка.
* Удаление строки из списка по индексу.
* Отображение всех строк, сохранённых в списке.
* Загрузка данных из текстового файла и их сохранение в бинарный файл.
* Сортировка строк списка по длине методом Шелла.
* Полная очистка списка с обнулением содержимого бинарного файла.

Проведён анализ производительности программы при загрузке строк из текстового файла. Тестирование показало, что время выполнения операций увеличивается с ростом объёма данных, что соответствует ожидаемому поведению для структур данных, работающих с файлами.

Разработанная программа может быть использована в задачах, требующих работы с большими объёмами данных, хранимых в бинарном формате. Она продемонстрировала устойчивость и корректность выполнения операций над двусвязным списком.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Прата С.; Язык программирования C++ : лекции и упражнения: [пер. с англ.] / Стивен Прата. - Москва [и др.] : Вильямс, 2012. - 1244 с. : ил. ; 25 см. - Предм. указ.: с. 1240-1244. - Пер. изд.: C++Primer Plus / S. Prata. - 1000 экз. - ISBN 978-5-8459-1778-2 (в пер.)
2. Мейерс С.; Эффективный и современный С++ : 42 рекомендации по использованию С++11 и С++14 : [пер. с англ.] / Скотт Мейерс. - Москва [и др.] : Вильямс, 2016. - 303 с. ; 25 см. - Предм. указ.: с. 301-303. - Пер. изд.: Effective Modern C++ / Scott Meyers.

# ПРИЛОЖЕНИЕ

**Листинг кода программы:**

**main.cpp:**

#include "linked\_file.h"  
#include <iostream>  
#include <string>  
  
using namespace std;  
  
int main() {  
 LinkedFile file("linked\_list.bin");  
 int choice, index;  
 string input;  
  
 do {  
 cout << "\n1. Add string" << endl;  
 cout << "2. Display list" << endl;  
 cout << "3. Insert string at index" << endl;  
 cout << "4. Sort list" << endl;  
 cout << "5. Delete stroke" << endl;  
 cout << "6. Display node" << endl;  
 cout << "7. Clear file and list" << endl;  
 cout << "8. Load from text file" << endl;  
 cout << "9. Save to text file" << endl;  
 cout << "10. Save to binary file" << endl;  
 cout << "11. Load from binary file" << endl;  
 cout << "0. Exit" << endl;  
 cout << "Choose option:";  
  
 while (true) {  
 cin >> choice;  
 if (cin.fail() || choice < 0 || choice > 11) {  
 cin.clear();  
 cin.ignore(1000, '\n');  
 cout << "Invalid input. Try again: ";  
 } else {  
 cin.ignore(1000, '\n');  
 break;  
 }  
 }  
  
 switch (choice) {  
 case 1:  
 cout << "Enter string: ";  
 getline(cin, input);  
 file.add(input);  
 break;  
 case 2:  
 file.display();  
 break;  
 case 3:  
 cout << "Enter index: ";  
 cin >> index;  
 cin.ignore(1000, '\n');  
 cout << "Enter string: ";  
 getline(cin, input);  
 file.insertAt(index, input);  
 break;  
 case 4:  
 file.shellSort();  
 break;  
 case 5:  
 cout << "Enter index to delete: ";  
 cin >> index;  
 cin.ignore(1000, '\n');  
 file.deleteNode(index);  
 break;  
 case 6:  
 cout << "Enter index to get string: ";  
 cin >> index;  
 cin.ignore(1000, '\n');  
 file.getStringAt(index);  
 break;  
 case 7:  
 file.clear();  
 break;  
 case 8:  
 file.clear();  
 cout << "Enter text file name to load from: ";  
 getline(cin, input);  
 file.loadFromTextFile(input);  
 break;  
 case 9:  
 cout << "Enter text file name to save to: ";  
 getline(cin, input);  
 file.saveToTextFile(input);  
 break;  
 case 10:  
 cout << "Successfuly saved" << endl;  
 break;  
 case 11:  
 file.display();  
 break;  
 case 0:  
 cout << "Exit...\n";  
 break;  
 default:  
 cout << "Incorrect choice.\n";  
 }  
 } while (choice != 0);  
  
 return 0;  
}

**linked\_file.h:**

#ifndef LINKED\_FILE\_H  
#define LINKED\_FILE\_H  
  
#include <fstream>  
#include <string>  
#include <iostream>  
#include <vector>  
#include <fstream>  
  
using namespace std;  
  
class LinkedFile : public std::fstream {  
private:  
 struct Header {  
 streampos head;  
 int count;  
 };  
 struct Node {  
 streampos next;  
 size\_t length;  
 };  
 Header header;  
 string filename;  
 void writeHeader();  
 void readHeader();  
public:  
 LinkedFile(const std::string& filename);  
 void add(const std::string& str);  
 void display();  
 void insertAt(int index, const std::string& str);  
 void shellSort();  
 void deleteNode(int index);  
 void getStringAt(int index);  
 void clear();  
 void loadFromTextFile(const string &textFilename);  
 void saveToTextFile(const string &textFilename);  
 void saveToBinaryFile(const string &binaryFilename);  
 void loadFromBinaryFile(const string &binaryFilename);  
};  
  
#endif

**linked\_file.cpp:**

#include "linked\_file.h"  
#include <vector>  
#include <string>  
#include <iostream>  
#include <fstream>  
  
// Конструктор  
LinkedFile::LinkedFile(const string& filename) : filename(filename) {  
 this->open(filename, ios::in | ios::out | ios::binary);  
 if (!this->is\_open()) {  
 this->open(filename, ios::out | ios::binary);  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
 } else {  
 readHeader();  
 }  
}  
  
  
 // Запись заголовка в начало файла  
 void LinkedFile::writeHeader() {  
 this->seekp(0, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&header.head), sizeof(header.head));  
 this->write(reinterpret\_cast<char\*>(&header.count), sizeof(header.count));  
 }  
  
// Чтение заголовка из файла  
void LinkedFile::readHeader() {  
 this->seekg(0, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&header.head), sizeof(header.head));  
 this->read(reinterpret\_cast<char\*>(&header.count), sizeof(header.count));  
}  
  
// Добавление строки в конец списка  
void LinkedFile::add(const string& str) {  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
  
 // Переход в конец файла  
 this->seekp(0, end);  
 streampos currentPos = this->tellp();  
  
 // Запись нового узла  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 // Обновление предыдущего узла, если он есть  
 if (header.head == -1) {  
 header.head = currentPos;  
 } else {  
 streampos prevPos = header.head;  
 Node prevNode;  
  
 while (true) {  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
 if (prevNode.next == -1) {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentPos), sizeof(currentPos));  
 break;  
 }  
 prevPos = prevNode.next;  
 }  
 }  
  
 header.count++;  
 writeHeader();  
}  
  
// Отображение всех строк  
void LinkedFile::display() {  
 if (header.head == -1) {  
 cout << "List is empty.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
 int counter = 0;  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 counter++;  
 cout << counter << ": " << str << "\n";  
  
 currentPos = node.next;  
 }  
}  
  
// Вставка строки по индексу  
void LinkedFile::insertAt(int index, const string& str) {  
 if (cin.fail()) {  
 throw invalid\_argument("Invalid input");  
 }  
 if (index < 0) {  
 throw out\_of\_range("Index cannot be negative.");  
 }  
 if (index > header.count) {  
 throw out\_of\_range("Index is out of range.");  
 }  
  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
  
 // Определение позиции нового узла  
 streampos newPos = this->tellp();  
 this->seekp(0, ios::end);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 if (header.count == 0) {  
 // Вставка в пустой список  
 header.head = newPos;  
 cout << "Inserted first node.\n";  
 } else if (index == 0) {  
 // Вставка в начало списка  
 newNode.next = header.head;  
 header.head = newPos;  
 this->seekp(newPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 cout << "Inserted node at the beginning.\n";  
 } else {  
 // Вставка в середину или конец списка  
 streampos prevPos = header.head;  
 Node prevNode;  
  
 for (int i = 0; i < index - 1; i++) {  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
 prevPos = prevNode.next;  
 }  
  
 // Считывание предыдущего узла  
 this->seekg(prevPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
  
 newNode.next = prevNode.next;  
 prevNode.next = newPos;  
  
 // Запись обновлённого предыдущего узла  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&prevNode.next), sizeof(prevNode.next));  
  
 // Запись нового узла  
 this->seekp(newPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
  
 cout << "Inserted node at index " << index << ".\n";  
 }  
  
 header.count++;  
 writeHeader();  
}  
  
void LinkedFile::deleteNode(int index) {  
 if (index < 0 || index >= header.count) {  
 cout << "Out of bounds.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 streampos prevPos = -1;  
 Node currentNode;  
  
 for (int i = 0; i <= index; i++) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&currentNode.next), sizeof(currentNode.next));  
 this->read(reinterpret\_cast<char\*>(&currentNode.length), sizeof(currentNode.length));  
  
 if (i == index) {  
 if (prevPos == -1) {  
 header.head = currentNode.next;  
 } else {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentNode.next), sizeof(currentNode.next));  
 }  
  
 header.count--;  
 writeHeader();  
 cout << "Node deleted.\n";  
 return;  
 }  
  
 prevPos = currentPos;  
 currentPos = currentNode.next;  
 }  
}  
  
 void LinkedFile::getStringAt(int index) {  
 if (index < 0 || index >= header.count) {  
 cout << "Out of bounds.\n";  
 return;  
 }  
  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
  
 for (int i = 0; i <= index; i++) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 if (i == index) {  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 cout << "String at index " << index << ": " << str << "\n";  
 return;  
 }  
  
 currentPos = node.next;  
 }  
 }  
  
void LinkedFile::clear() {  
 this->close();  
 this->open(filename, ios::out | ios::binary);  
 if (this->is\_open()) {  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 cout << "File and list cleared.\n";  
 } else {  
 cout << "Failed to clear the file.\n";  
 }  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
}  
  
void LinkedFile::shellSort() {  
 if (header.count <= 1) {  
 cout << "List is already sorted or empty.\n";  
 return;  
 }  
  
 // Считывание всех узлов в память  
 vector<string> strings; // Хранение строк  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 strings.push\_back(str);  
  
 currentPos = node.next;  
 }  
 int n = strings.size();  
 for (int gap = n / 2; gap > 0; gap /= 2) {  
 for (int i = gap; i < n; i++) {  
 string temp = strings[i];  
 int j;  
 for (j = i; j >= gap && strings[j - gap].size() > temp.size(); j -= gap) {  
 strings[j] = strings[j - gap];  
 }  
 strings[j] = temp;  
 }  
 }  
 this->close();  
 this->open(filename, ios::out | ios::binary);  
 header.head = -1;  
 header.count = 0;  
 writeHeader();  
 streampos prevPos = -1;  
 for (const auto& str : strings) {  
 // Позиция текущего узла  
 Node newNode;  
 newNode.length = str.size();  
 newNode.next = -1;  
 this->seekp(0, ios::end);  
 streampos currentPos = this->tellp();  
 // Если первый узел, обновляем заголовок  
 if (header.head == -1) {  
 header.head = currentPos;  
 }  
  
 // Связывание предыдущего узла с текущим  
 if (prevPos != -1) {  
 this->seekp(prevPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&currentPos), sizeof(currentPos));  
 }  
  
 // Запись текущего узла  
 this->seekp(currentPos, ios::beg);  
 this->write(reinterpret\_cast<char\*>(&newNode.next), sizeof(newNode.next));  
 this->write(reinterpret\_cast<char\*>(&newNode.length), sizeof(newNode.length));  
 this->write(str.c\_str(), newNode.length);  
  
 prevPos = currentPos;  
 header.count++;  
 }  
 writeHeader();  
 this->close();  
 this->open(filename, ios::in | ios::out | ios::binary);  
 cout << "List has been sorted by string length.\n";  
}  
  
// Загрузка списка из текстового файла  
void LinkedFile::loadFromTextFile(const string& textFilename) {  
 ifstream inFile(textFilename, ios::in);  
  
 if (!inFile) {  
 cerr << "Error: Unable to open file " << textFilename << endl;  
 return;  
 }  
 string fileContent((istreambuf\_iterator<char>(inFile)), istreambuf\_iterator<char>());  
 inFile.close();  
 string currentLine;  
 for (char c : fileContent) {  
 if (c == '\n') {  
 add(currentLine);  
 currentLine.clear();  
 } else {  
 currentLine += c;  
 }  
 }  
 if (!currentLine.empty()) {  
 add(currentLine);  
 }  
 cout << "File loaded successfully from " << textFilename << endl;  
}  
  
 // Сохранение списка в текстовый файл  
 void LinkedFile::saveToTextFile(const string& textFilename) {  
 ofstream textFile(textFilename);  
 if (!textFile.is\_open()) {  
 cerr << "Failed to open text file.\n";  
 return;  
 }  
 streampos currentPos = header.head;  
 Node node;  
 string str;  
 while (currentPos != -1) {  
 this->seekg(currentPos, ios::beg);  
 this->read(reinterpret\_cast<char\*>(&node.next), sizeof(node.next));  
 this->read(reinterpret\_cast<char\*>(&node.length), sizeof(node.length));  
 str.resize(node.length);  
 this->read(&str[0], node.length);  
 textFile << str << "\n";  
 currentPos = node.next;  
 }  
 cout << "Saved data from binary file to text file.\n";  
 }  
  
// Сохранение списка в бинарный файл  
void LinkedFile::saveToBinaryFile(const string &binaryFilename) {  
 ofstream outFile(binaryFilename, ios::binary);  
 if (!outFile) {  
 cerr << "Error: Unable to open file for writing: " << binaryFilename << endl;  
 return;  
 }  
 outFile.write(reinterpret\_cast<const char\*>(&header), sizeof(header));  
 streampos currentPos = header.head;  
 while (currentPos != streampos(-1)) {  
 Node node;  
 this->seekg(currentPos);  
 this->read(reinterpret\_cast<char\*>(&node), sizeof(Node));  
 outFile.write(reinterpret\_cast<const char\*>(&node), sizeof(Node));  
 string data(node.length, '\0');  
 this->read(&data[0], node.length);  
 outFile.write(data.c\_str(), node.length);  
 currentPos = node.next;  
 }  
 outFile.close();  
 cout << "Linked list successfully saved to binary file: " << binaryFilename << endl;  
}  
  
// Загрузка списка из бинарного файла  
void LinkedFile::loadFromBinaryFile(const string& binaryFilename) {  
 ifstream inFile(binaryFilename, ios::binary);  
 if (!inFile) {  
 cerr << "Error: Unable to open file for reading: " << binaryFilename << endl;  
 return;  
 }  
 clear();  
 while (!inFile.eof()) {  
 size\_t length = 0;  
 inFile.read(reinterpret\_cast<char\*>(&length), sizeof(length));  
 if (inFile.eof()) break;  
 string data(length, '\0');  
 inFile.read(&data[0], length);  
 add(data);  
 }  
 inFile.close();  
 cout << "List successfully loaded from binary file: " << binaryFilename << endl;  
}